N94-35056

Automated Analysis of Complex Data

Robert St. Amant and Paul R. Cohen
Computer Science Department
University of Massachusetts
Amherst, MA 01003
stamant@cs.umass.edu, cohen@cs.umass.edu

1. Introduction

Igor is a knowledge-based system designed for exploratory statistical analysis of complex systems and environments. Igor has two related goals: to help automate the search for interesting patterns in data sets, and to help develop models that capture significant relationships in the data. Igor supports and complements the efforts of a human analyst in examining large or complex data sets [11].

Igor explores a data set with the statistical operations of exploratory data analysis (EDA). Examples of EDA operations are univariate power transforms, bivariate relationship partitioning to distinguish separate effects, and analysis of patterns in residuals. Through these operations Igor incrementally constructs partial descriptions or models of data. When interesting patterns are observed in a model or its residuals in the data, Igor opportunistically selects appropriate operations to explore the new phenomena. As the analysis proceeds, a more complete picture of the data set gradually emerges.

We must consider many different tradeoffs in designing a system for statistical reasoning: operator generality versus power, language expressiveness versus efficiency, opportunism versus control. Of these the key tradeoff is between opportunism and control. EDA operations are powerful: they can be applied in many situations in different combinations, producing results whose proper interpretation depends on context. Effective analysis depends on taking advantage of new results to guide search in appropriate directions [7].

In terms of representation, we must decide how primitive statistical operators should be combined into higher level ones, and how these operators should interact with each other. If EDA operations are implemented as procedures that call other statistical procedures, as in a programming language, we have strict control over the results we produce, but little flexibility in responding to unexpected findings and applying appropriate context-specific techniques. If EDA operations are implemented as rules that fire whenever "interesting" intermediate results appear, on the other hand, we have the necessary opportunism, but find it difficult to capture strategic aspects of analysis.

We can approach this problem with the techniques of opportunistic planning. We have developed in Igor a planning language that balances control and opportunism. By explicit representation of the goals of exploratory analysis, we can take advantage of strategic knowledge about data analysis to structure and reduce search. Monitoring structures opportunistically detect intermediate and end results with interesting characteristics. The

This research is supported by the Advanced Research Projects Agency and Rome Laboratory under contracts F30602-91-C-0076 and F30602-93-C-0100.

PRECEDING PAGE BLANK NOT FILMED

plan and goal representation lets Igor select appropriate context-dependent sequences of operations during the analysis.

Igor is intended for robust assistance in domains with an enormous search space of hypotheses. Conventional packages offer statistical tests and an environment in which the user may either program scripts or run through an analysis by hand. Igor allows more flexible interaction. An analyst may apply a variety of heterogeneous strategies to produce and confirm results. Some processes may be scripted to proceed without interaction, while others may be defined to incorporate human guidance.

Our work has dealt mainly with analyzing the behavior of artificial intelligence programs. Because an AI system may react in complex ways to the influences of its environment, the reasons for its behavior may not be obvious from execution traces. This is a challenging domain for Igor. We have also looked into automating the analysis of Landsat wildlife habitat data, but we have gone no farther than designing a few basic procedures for data preparation [8].

In the next section we discuss elementary strategies for exploring data. In Section 3 we outline the planning representation, the relevance of planning to statistical analysis and how the representation supports the process. Section 4 describes an example of Igor in practice. Section 5 concludes with a discussion of the benefits of the general approach and plans for future work.

2. Strategies for Exploring Data

Exploratory studies are the informal prelude to experiments, in which questions and procedures are refined. Exploration is like working in a test kitchen: before one writes down the final version of a recipe, one first tries out possible alternative procedures and evaluates the results. Exploratory results influence confirmatory studies in a cycle of successively more refined exploration and confirmation [2, 5, 6, 12].

We apply two general strategies in exploring data: one generates simplifying descriptions of data, the other extends and refines surface descriptions of data. We simplify data by constructing partial descriptions and models that capture particular characteristics of the data. The descriptions range from simple summary statistics, such as means and medians, to complex domain models. We make descriptions more effective by looking beyond surface descriptions at what is left unexplained. For example, a regression line may be a good description of the general trend in a relationship, but an analysis of residuals can give an entirely different picture at a lower level of detail. Exploratory strategies generate increasingly detailed, complementary descriptions of data.

EDA strategies often takes advantage of intermediate results that suggest further areas of exploration. We can best illustrate the process with a brief example, adapted from Tukey [12]. Consider Figure 1, which plots the population of the U.S. between 1800 and 1950.

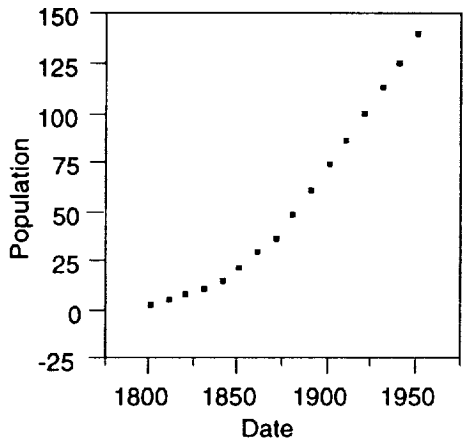

Figure 1: Population vs Date

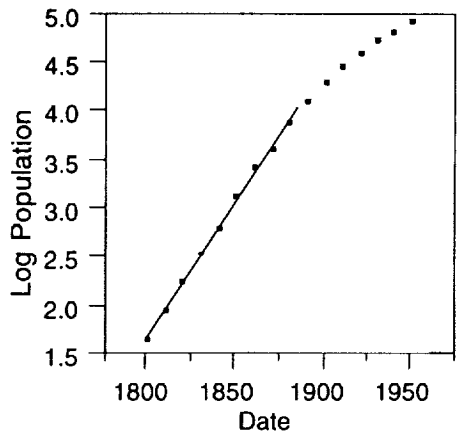

Figure 2: Log Population vs Date

We first notice that population in earlier decades increases at a slower rate than in later years. We can fit a small-degree polynomial such as a quadratic to the data, or, equivalently, transform each point by taking its square root. This kind of transformation serves to straighten the data points so that further structure may be observed. What we find, however, is that the points are not collinear after the transformation. A pattern in the residuals tells us that there is further structure to be elucidated. We thus return to the original data scale to try a different approach.

While the earlier data points seem to curve upward, the later points, from about 1900, seem to have constant slope. We thus partition the relationship at this point, and analyze each partition separately. It turns out that higher order curves fit the later partition little better than a straight line. From the residuals of this fit we may be able to extract further information--for example, that population took a dip in the decade of 1940. The linear fit and residuals, which we interpret as descriptions of trend and local detail, together form a satisfactory model of the later partition. For the earlier partition we first apply a log transform to straighten the data points, as in Figure 2, and then proceed as with the later partition.

Taken together, we now have a reasonably detailed model of population during these years. It shows that population increased exponentially between 1800 and 1900, but then slowed to a linear rate. Local effects can be seen in the residuals of the trend descriptions. Our construction of this composite model was guided by observation of an inflection point in the data, and the realization that the first, simpler model was not sufficient to capture significant characteristics of the data.

Given a basic set of statistical tools, humans can easily follow similar reasoning. The task is considerably more difficult for a computer. Our description has neglected much of the complexity of the decisions involved. Many intermediate results arise, but only a few of them turn out to be interesting. In general there are always choices to be made about which techniques best suit the situation and which results are most interesting to pursue.

When we explore data effectively we exploit a tension between opportunism and control. Opportunism lets us explore new interesting results that arise unpredictably. Control determines which results are promising, how they might best be evaluated, and when particular paths of reasoning might be abandoned. Without a proper balance between opportunism and control, an automated system may be unable to make the simplest discoveries, or may face an explosive search space.

3. A Planning Language for Data Exploration

In the foregoing example we incrementally built a structured interpretation, using weak heuristics, from raw data. We can view the process as a form of planning. In planning, sequences of simple operations are combined for complex effect. Planning may be incremental and opportunistic, based on constructing and revising plans according to information acquired during the process. The analogy between planning and data exploration extends further:

- Exploratory strategies are plans consisting of sequences of statistical operations; these operations are actions that transform data relationships.

- As in planning, primitive exploratory operations can be combined in different ways for different effects. For example, in considering a relationship between two variables, it makes a great difference whether we remove outliers before or after applying a transformation to the relationship.

- Conversely, abstract statistical operations often decompose naturally into more primitive operations, just as in hierarchical plan decomposition. For example, the abstract action of fitting a robust line to a relationship may expand to partitioning the relationship, removing outliers, calculating medians, and combining the results.

- Selection of the most effective strategy is akin to selection of an appropriate plan to satisfy a given goal. We must often evaluate different paths to find the most effective one.

- Just as plans fail, the results of an exploratory operation may not be useful, requiring repeated application perhaps with different parameters. Retrying an operation is analogous to retrying an action as a part of plan failure recovery. Selecting a different, more promising strategy corresponds to replanning.

Using this analogy we have developed a planning language for Igor, based on the RESUN signal interpretation system [1]. RESUN is a blackboard-based control planner which supports goal satisfaction through flexible combination of scripts and actions. Opportunism is managed by mechanisms that monitor the state of plans and by heuristics that determine when and how particular goals are satisfied.

At the lowest level, the data structures manipulated in the planning representation are frames. A variable is a simple frame; a linear relationship between two variables is a slightly more complex hierarchy of frames; an annotated causal model is a highly interconnected hierarchy of frames. Igor allows specialization of two built-in data types – *basic sequence* and *basic relationship*. We call frames and hierarchies of all types *structures*.

The primitive operations provided by the representation are called *actions*. An action is a data transformation or decomposition of an input structure to an output structure. A log transform is a simple example of an action; it applies a log function to each element in a sequence and collects the results. More complex transformations include smoothing, outlier removal, and fit operations. Each action has an associated goal form and may be triggered by the establishment of a matching goal. The definition of the log transform action is shown in Figure 3.

```
(define-action  (log-transform-action
          :goal  (log-transform-goal  ?variable  ?result)
          :input  (variable)
          :output  (result))
    :action  (log-variable  variable))
```

Figure 3: Action form for log transform

Actions are combined in *scripts*. A script is a sequence of subgoals whose associated actions transform one structure into a more concise, better parametrized, more descriptive structure. Scripts, like actions, have associated goal forms, and thus may be combined hierarchically to satisfy the goals of other scripts. Combination of subgoals in a script is governed by the *specification* of the script. A script specification defines how its subgoals must be satisfied in order for the top level goal to be satisfied. Specification constructs allow sequential combination of subgoals, iteration over sets of subgoals, conditionalization on tests of variable values, and activation of subgoals in parallel. The example script in Figure 4 combines a conditional test of a variable with a sequence of subgoals. This script finds discontinuities in a variable by examining the differences between contiguous elements of the sorted variable values, in a simple form of cluster analysis. Relatively large steps indicate breaks between groups of values. Each subgoal in the specification transforms an intermediate result, producing in the end a variable in which elements are mapped to the clusters in which they fall.

```
(define-script  find-variable-discontinuities-script
    :goal (explore variable ?variable ?context)
    :input (variable context)
    :output  (transformed-variable)
    :spec (:COND (variable)
                 (continuous-valued-variable-p  variable)
                 (:SEQ
                       sort-goal
                       difference-goal
                       outlier-positions-goal
                       outlier-position-values-goal
                       transform-by-discontinuities-goal)))
```

Figure 4: Script form for variable discontinuities

Scripts and actions control procedural execution in the representation, managed flexibly by goal establishment. These constructs still do not provide the degree of opportunism and context-specific control we associate with exploration, however. For this we rely on an explicit representation of context, and two mechanisms that depend on context, *monitoring* and *focusing*.

Execution of an action causes a new structure to be generated. When actions execute sequentially as part of a script, these new structures may be stored as intermediate results of the script. A *context* structure is simply the sequence of intermediate and end results produced by execution of a script. We associate a context with a script and the goal which

the script satisfies. Because script subgoals may be satisfied by other scripts, hierarchical structures of contexts may be built up during exploration. These structures provide contextual information to monitoring and focusing mechanisms.

A strictly goal-driven system can find it difficult to take new structures under consideration during the search process. A *monitor* is a special script that is activated automatically when an object is created as an intermediate or end result of the execution of a script. A monitor establishes *exploration goals* for the new result and makes non-local updates to the context hierarchy. Monitors evaluate the 'interestingness' of results, taking context information into account, to see which kinds of exploration, and hence goals, are relevant. Domain-specific as well as general heuristic knowledge can influence the selection of appropriate goals by a monitor.

Focusing heuristics guide the exploration process based on local context information. Focusing heuristics are activated whenever there is a choice between which goals to pursue and which scripts to apply; they evaluate the precedence of active goals and the relevance of matching scripts when deciding which scripts should be activated and which ignored. We use focusing heuristics to evaluate the cost of pursuing particular search paths. A focusing heuristic is free to prune the goals or scripts it takes as input, temporarily or permanently. As with monitors, a focusing heuristic may take advantage of domain-specific knowledge in its processing.

4. An Igor Example

In this section we examine Igor's analysis of the results of an experiment with Phoenix, a simulated environment populated by autonomous agents.

Phoenix simulates forest fires in Yellowstone National Park and the agents that fight the fires. Agents include watchtowers, fuel trucks, helicopters, bulldozers, and, coordinating the efforts of all, a fireboss. Fires burn in unpredictable ways due to wind speed and direction, terrain and elevation, ground cover type and natural boundaries such as rivers, roads, and lakes. Agents behave unpredictably as well, because they instantiate plans as they proceed, reacting to immediate, local situation changes. Phoenix is a complex simulation; even when the agents are successful in containing a fire we may find it difficult to explain why particular behavior is effective. We thus run experiments in which we control and test specific aspects of Phoenix.

In one experiment we examined the relationship between the thinking speed of the fireboss and the rate of environmental change. Thinking speed is controlled by the Real-Time Knob, or RTK, which sets the ratio of CPU time in the fireboss to simulation time in the environment. Thus, for example, during the development of Phoenix the value of RTK was fixed at 1.0, which corresponds to five minutes of simulation time elapsed per one second of CPU time. By setting the value of RTK higher or lower we change how quickly the fireboss can react to external events and build plans to deal with them. Environmental change in this experiment is influenced by wind speed. When wind speed is low, fire spread is slow and predictable. At higher wind speeds fire spreads more quickly and takes less predictable paths over the ground cover.

We created a fire fighting scenario to be fought by the fireboss and four bulldozers. We ran 385 trials using the same basic scenario but setting wind speed and RTK to different values at the beginning of each trial. During each trial we collected some forty measurements, including Wind Speed, RTK, Success, and Area Burned. Our example will cover these variables (Table 1.)

Table 1: Phoenix parameters

Variable	Abbr	Values
Wind Speed	WS	3, 6, 9 km/hr
Real-Time Knob	RTK	0.333, 0.454, 0.555, 0.714, 1.000, 1.666
Success	Success	0, 1
Area Burned	Area	continuous values

First we describe the steps Igor takes in analyzing this data set, and then we summarize with a broader picture of the exploration process. Our discussion will focus on exploration of the relationship between RTK and Success, or RTK-S.

Igor begins its analysis with a single goal, explore-data. The top level script that matches this goal, explore-data-script, expands into parallel subgoals, explore-variables and explore-relationships. These subgoals are satisfied by scripts which expand into goals for the exploration of each variable individually and relationships between them.

Exploration of single variables involves scripts for straightening variables through power transforms, detecting discontinuities and regions of constancy in variable values, finding possible clusters, and calculating standard summary statistics. Exploration of relationships includes 'untilting' skewed relationships with power transforms, mapping categorical variable relationships into contingency tables, and applying linear fit functions and testing residuals. On detecting a particular characteristic in a structure, a script most often produces a transformation of the structure into a form in which the characteristic is easily seen and manipulated. These scripts are not applied in linear fashion, but are rather selected by goal matching, pruned and ordered by focusing heuristics, and re-applied by monitors.

An interesting avenue is taken in exploring how the probability of success depends on thinking speed. The process interests us because it deals with a high level description of the relationship between two primary variables in the experiment, RTK and Success. One of the scripts activated by the explore-relationship goal is the discrete-vs-mean-transformation. When this script determines that RTK has discrete values, it proceeds by partitioning Success by the values of RTK, reducing RTK to its six distinct values, and mapping the statistic mean over each of the Success partitions. (Because Success has binary values, 0 for failure and 1 for success, the mean of a Success partition is equivalent to the proportion of successes in that partition.) The resulting sequences, containing the reduced values of RTK and the mean values of Success per partition, are associated in a new relationship, RTK-pS. We can interpret this relationship as the probability of succeeding given a particular value of RTK. Graphically the relationship RTK-pS looks like the plot in Figure 5.

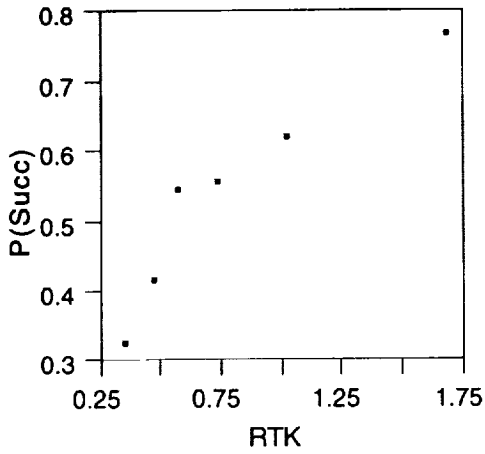

Figure 5: RTK-pS relationship

The result of the discrete-vs-mean-transformation is a transformed relationship. A monitor at the explore-relationships level detects the creation of the RTK-pS relationship and adds it to the context structure for the explore-relationships goal. It can then be explored in turn.

One of the exploration goals generated for RTK-pS is the sequence-fit goal, which is matched by the linear-fit script. The linear-fit script fits a line to the relationship, generating both a simple regression equation and a sequence of residuals. The residuals sequence is detected by a monitor and examined for patterns. The simplest pattern-detecting script finds that the residuals of RTK-pS are bitonic, increasing then decreasing. A monitor makes this available in the context of the exploration goal of RTK-pS.

Now the activation of scripts to match the sequence-fit goal is controlled by a focusing heuristic. If the linear-fit script produces a good enough fit (i.e., with enough variance accounted for,) and there is no structure in the residuals, then the goal is satisfied. Here, however, because of the residuals, the focusing heuristic must continue the search. There are two further possibilities available to the focusing heuristic. The first is to produce a transform for RTK-pS and retry the linear-fit script. The second is to partition the relationship at or near the inflection point in the residuals, and generate a fit for each partition separately during exploration.

Both of these approaches are plausible. Because we know that the mean of Success can never rise above 1.0, we could fit a sublinear function to the relationship, with asymptote at 1.0, as shown in Figure 6. This gives a better fit than a straight line, but still not a perfect one. Alternatively we recall that the initial setting of RTK is 1.0, the setting at which the system was designed and tested. We observe that the discontinuity in RTK-pS occurs near this value. Thus we may be seeing not a continuous relationship between the variables but two separate modes of operation, one at low thinking speed and the other at high thinking speed. Within each mode the RTK-pS relationship is linear. This interpretation is given in Figure 7.

Neither of these competing interpretations clearly rules out the other. If we believe that the design of Phoenix provides smooth degradation in performance as thinking speed decreases, we might prefer the first interpretation. The second interpretation is also plausible, except that degradation is not smooth after some threshold value of RTK is

reached. The additional evidence that this threshold occurs near the RTK setting of 1.0, possibly indicating that Phoenix plans rely implicitly on this setting, inclines us to credit the second interpretation.

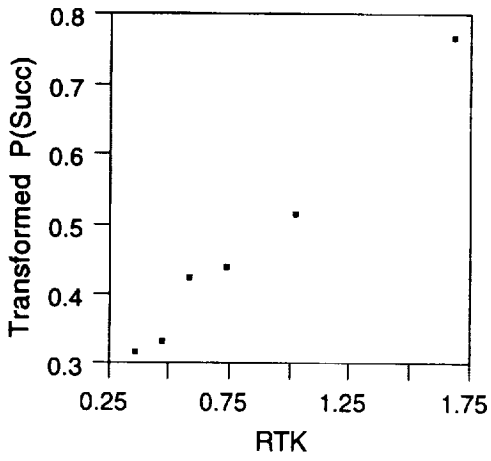

Figure 6: Straightened RTK-pS

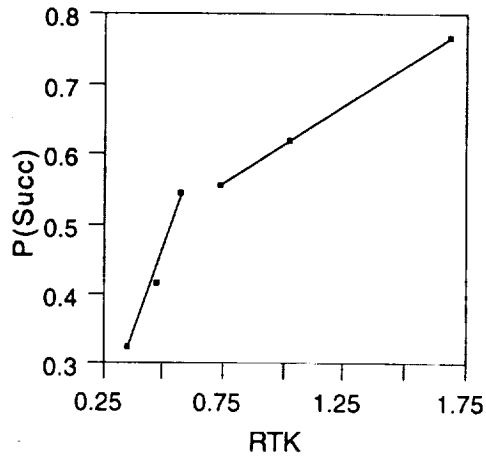

Figure 7: Partitioned model of RTK-pS

The focusing heuristic can clearly take advantage of such domain-dependent knowledge. In this case, however, we have taken a simpler approach. Applying a transformation to RTK-pS is relatively inexpensive, in terms of increase in the size of the search space. Partitioning the relationship, on the other hand, is a great deal more expensive. In general when partitioning Igor must consider the partitions separately, and in some cases recombine the individual results. Thus the focusing heuristic runs the sequence-fit scripts in sequence, rather than in parallel, and will not proceed to the later ones if the initial inexpensive ones succeed.

Finally we present an overview of Igor's processing. Figure 8 traces significant points in plan and goal instantiation during the exploration process.

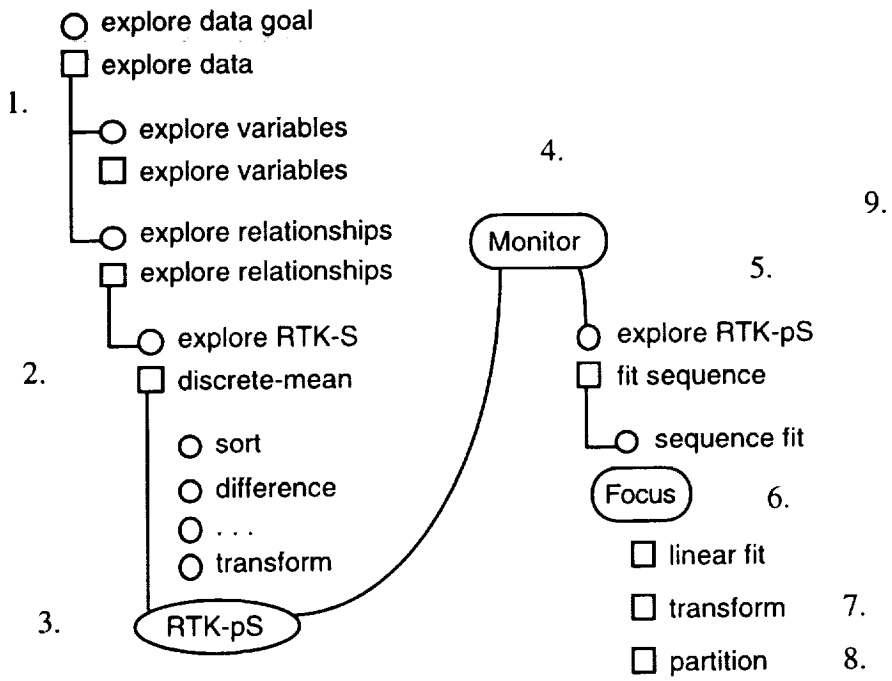

Figure 8: Exploration trace

1. Exploration goals are established for variables and relationships.
2. Relationship RTK-S is explored, triggering the discrete-vs-mean-transformation.
3. Relationship RTK-pS is generated.
4. A monitor detects RTK-pS and establishes an exploration goal for the relationship.
5. A sequence-fit goal is established for RTK-pS.
6. A focusing heuristic determines that a linear fit is appropriate.
7. Residual patterns cause a nonlinear fit to be tried.
8. Inadequate nonlinear fit causes a partitioning of the relationship.
9. Further exploration on the partitions occurs.

5. Conclusions

We have examined some of the issues involved in automating exploratory data analysis, in particular the tradeoff between control and opportunism. We have proposed an opportunistic planning solution for this tradeoff, and have implemented a prototype, Igor, to test the approach. Our experience in developing Igor has been surprisingly smooth. In contrast to earlier versions that relied on rule representation, it has been straightforward to increment Igor's knowledge base without causing the search space to explode. The planning representation appears to be both general and powerful, with high level strategic knowledge provided by goals and plans, and hooks for domain-specific knowledge provided by monitors and focusing heuristics.

Our future plans include incorporating detailed domain-specific knowledge into Igor, which will let us explore the interaction between general strategic knowledge and domain-specific control knowledge [8, 9, 10]. A domain we find especially interesting is experimental results describing the behavior of AI systems such as planners and schedulers.

We also wish to examine the relationship between exploratory data analysis and automated causal modeling. Some of the results produced by EDA appear to be useful cues to humans in developing causal models. We may be able to exploit such results in an automated system as well. Further, a causal model may be able to give search guidance to Igor, for example in distinguishing between proximal and distal causes of an observed effect [3, 4].

Our final concern is with evaluation. How sensitive is Igor to its parameter settings? How much information does Igor require to draw conclusions about a particular effect? Are the chains of statistical reasoning Igor produces coherent to humans? We will explore these and other questions in future work.

Acknowledgements

This research is supported by the Advanced Research Projects Agency and Rome Laboratory under contracts F30602-91-C-0076 and F30602-93-C-0100. The US Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon.

References

[1] Carver, N., & Lesser, V., 1993. A Planner for the Control of Problem-Solving Systems. *IEEE Transactions on Systems, Man, and Cybernetics*, vol 23, no. 6.

[2] Cohen, P.R., 1993. *Empirical Methods for Artificial Intelligence*. MIT Press. Forthcoming.

[3] Cohen, P.R., Carlson, A., Ballesteros, L., St. Amant, R., 1993. Automating Path Analysis for Building Causal Models from Data. *Proceedings of the Ninth International Conference on Machine Learning*. Morgan Kaufmann.

[4] Cohen, P.R & Hart, D.M., 1993. Path analysis models of an autonomous agent in a complex environment. *Proceedings of The Fourth International Workshop on AI and Statistics*. Ft. Lauderdale, FL.

[5] Gale, W.A., 1986. REX Review. In Gale, W.A. (Ed.), *Artificial Intelligence and Statistics*. Addison-Wesley.

[6] Hand, D.J., 1986. Patterns in Statistical Strategy. In Gale, W.A. (Ed.), *Artificial Intelligence and Statistics*. Addison-Wesley.

[7] Kulkarni, D., & Simon, H.A., 1990. Experimentation in Machine Discovery. In Schrager, J., & Langley, P., (Eds.), *Computational Models of Scientific Discovery and Theory Formation*. Morgan Kaufmann.

[8] Lansky, A.L, & Philpot, A.G., 1993. AI-Based Planning for Data Analysis Tasks. CAIA, IEEE Conference on AI Applications, Orlando, Florida.

[9] Langley, P., Bradshaw, G., & Simon, H.A., 1983. Rediscovering Chemistry with the BACON System. In R.S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine Learning An Artificial Intelligence Approach*. Morgan Kaufmann.

[10] Schrager, J., & Langley, P., 1990. Computational Approaches to Scientific Discovery. In Schrager, J., & Langley, P., (Eds.), *Computational Models of Scientific Discovery and Theory Formation*. Morgan Kaufmann.

[11] Silvey, P., 1993. IGOR: The MAD Scientist's Assistant or Building Models from Data. COINS Technical Report 93-??. University of Massachusetts, Amherst.

[12] Tukey, J.W., 1977. *Exploratory Data Analysis*. Addison-Wesley.